A Decomposition Approach for SMT-based Schedule Synthesis for Time-Triggered Networks

Francisco Pozo*, Wilfried Steiner†, Guillermo Rodriguez-Navas*, and Hans Hansson*
*School of Innovation, Design and Engineering, Mälardalen University
Västeras, Sweden
Email: {francisco.pozo, guillermo.rodriguez-navas, hans.hansson}@mdh.se
†TTTech Computertechnik AG
Vienna, Austria
Email: wilfried.steiner@tttech.com

***Abstract*—Real-time networks have tight communication latency and minimal jitter requirements. One way to ensure these requirements is the implementation of a static schedule, which defines the transmission points in time of time-triggered frames. Synthesizing such static schedules is known to be an NP-complete problem where the complexity is driven by the large number of constraints imposed by the network. Satisfiabily Modulo Theories (SMT) have been proven powerful tools to synthesize schedules of medium-to-large industrial networks. However, the schedules of new extremely large networks, such as integrated multi-machine factory networks, are defined by an extremely large number of constraints exceeding the capabilities of being synthesized by the tool alone.**

This paper presents a decomposition approach that will allow us to improve to synthesize schedules with up to two orders of magnitude in terms of the number of constraints that can be handled. We also present an implementation of a dependency tree on top of the decomposition approach to address application-imposed constraints between frames.

I. Introduction

A strategy to accomplish high reliability and a clear separation of the resources in time-triggered multihop networks is with the definition of an offline static schedule [1]. Offline schedules can be seen as a contract in which there is an agreement in the way the network resources are shared. In time-triggered communication, offline schedules define the points in time in which every frame will be sent through the links fulfilling a set of constraints. These constraints are defined by network requirements and can be formulated as a constraint problem in which a valid offline schedule is obtained when all constraints are satisfied. Synthesizing such offline schedules is a complex problem known to be NP-complete [2]. Specialized constraints solvers have been usually used to solve constraint problems [3], [4], but recently, satisfiability solvers have started to be applied for such synthesis problems [5]. Satisfiabiliy Modulo Theories (SMT) emerged to combine satisfiability solvers with decision procedures expressed in first-order logic with respect to background theories, allowing a much easier definition of the problem and high performance [6]. Two examples of state-of-the-art SMT solvers are Yices [7], which we select as our synthesizer, and Z3 [8].

SMT solvers have been successfully applied to synthesize medium-to-large time-triggered multihop networks [9], but these networks are steadily growing faster than the computation needed to synthesize their schedules. Extremely large networks, such as an integrated multi-machine network in an automation factory, are defined by a large amount of constraints that no general-purpose solver is capable to manage and solve in a reasonable amount of time on its own. The decomposition approach is a widely applied strategy to deal with large complex problems, dividing the overall problem in smaller subproblems that can be solved. Thus, we propose a first decomposition of network schedules by groups of frames, in which a group's schedule is synthesized independently of the others groups of frames and placed afterwards in the global schedule with a translation of the time values of every schedule. However in most cases, frames have dependency constraints between them, increasing the complexity of choosing the group of frames or with the probability of breaking dependency constraints between subproblems on the translation phase. We propose the implementation of a dependency tree between frames that will allow us choosing the desirable amount of frames per subproblem and guiding the translation of the schedule subproblems in a way that dependency constraints are not broken.

The main contribution of this paper is the decomposition approach for the SMT-based synthesis of schedules that allow us to schedule much larger time-triggered networks that also contain dependency constraints between frames. We evaluate the performance of our approach with different synthetic TTEthernet networks [10] and show that our approach is capable of speeding up the synthesis time three orders of magnitude compared to existing synthesizers. Increasing the performance of the synthesizer allows us to schedule larger networks, defined by up to two order of magnitudes more constraints. Also we evaluate the impact in the performance for different number of dependency constraints in regards to the synthesis time which only presents a small overhead thanks to the implementation of the dependency tree.

We introduce related work in SMT solving, schedule synthesis and decomposition approaches in Section II. We then present the basic terminology of time-triggered multihop networks, SMT-based synthesis and the definition of frame constraints in Section III. In Section IV we discuss two decomposition scheduling approaches. We evaluate the performance of our approaches in Section V. Finally, we conclude in Section VI.

978-1-4673-7929-8/15/$31.00

II. Related Work

SMT solvers have been applied to various applications: model checking [11], verification [12], automated test generation [13], synthesis of programs [14] and more. The first use of SMT solvers to synthesize schedules [9] has been done to schedule medium size time-triggered multihop networks. A similar approach has been used to co-synthesize a task and network schedule for time-triggered networked systems [15] and to synthesize time-triggered network-on-chip static schedules [16].

Different approaches have been studied to synthesize schedules besides SMT-solving. Specific algorithms have been developed to synthesize schedules for Profinet IO with a consumer-producer approach [17]. Different satisfiability solvers beside SMT have also been used, like SAT-solving for task and message scheduling on bus systems [18]. Constrained-based synthesis is also a similar approach to satisfiability, FlexRay bus schedules were synthesized using a Mixed Integer Programming approach [19]. Meta-heuristics have been also used as a common approach to solve complex problems, for example, a tabu-search has been performed to synthesize mixed-criticallity application schedules [20].

Decomposition approaches have been widely used to solve large complex problems that could not have been solved with traditional approaches and also to speed up the time needed to solve problems, a decomposition approach has been proposed to speed up the synthesis time of mixed systems [21]. Other work has been done to solve large optimization problems with the decomposition approach, some examples are, a steel plant schedule Mixed Integer Linear Programming problem was solved with decomposition [22], or a flow shop schedule combining a genetic algorithm with decomposition [23].

III. Basic Terminology

A. Network Definition

TTEthernet networks are mixed-criticality multihop networks composed of end systems, switches and communication links, in which time-triggered, rate-constraint, and best-effort traffic are combined on the same network. End systems are the producers and consumer of frames, and can only be connected to switches by communication links. However, switches can be connected to other switches and end systems. The physical topology of multihop networks is typically defined by an undirected graph $G(V, E)$ where V represent the end systems and switches of the networks, and E the communication links. TTEthernet is a bidirectional network, hence the communication links are bidirectional and logically defined by two directed unidirectional *dataflow links*, one for each direction. A path that connects a vertex sender to a vertex receiver is defined by a sequence of dataflow links declared as *dataflow path*. TTEthernet also allows the grouping of dataflow paths to permit a unique sender transmit frames to multiple receivers with the implementation of *virtual links*, defined in ARINC 664-p7 [24]. An example of a TTEthernet network can be seen in Figure 1 in which a virtual link with sender v_1 transmits the frame to two receivers v_6, v_7 with a group of two dataflow paths: $((v_1, v_4), (v_4, v_5), (v_5, v_6))$ and $((v_1, v_4), (v_4, v_5), (v_5, v_7))$.

In a frame being transmitted from a sender to one or multiple receivers in a virtual link resides a collection of *instances of frames*, where every instance of a frame is transmitted in exactly one of the dataflow link of the virtual link. As TTEthernet is a mixed-criticality network, there are three different classes of frames:

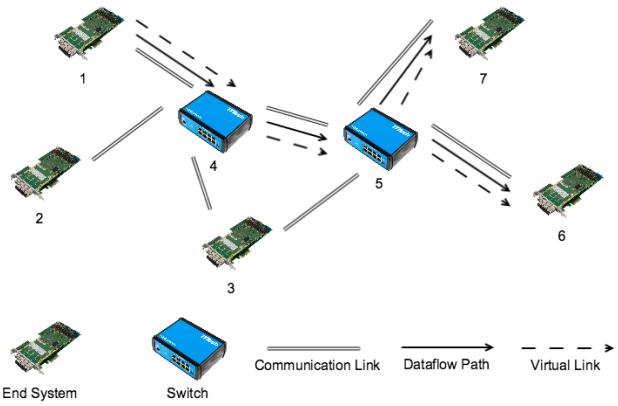

Fig. 1: TTEthernet Network example composed of two switches, five end systems and seven communication links

- Time-Triggered (TT): hard real-time requirements and follows an offline schedule.
- Rate-Constrained (RC): soft real-time requirements, guaranteed bandwidth with upper-bound latency.
- Best-Effort (BE): no real-time requirements, follows the standard Ethernet policy.

The offline schedule of a TTEthernet network contains all the instances of all the TT frames, which are defined by a triple $f^{(v_i,v_j)} = \{f.period, f.length, f^{(v_i,v_j)}.offset\}$. The period and length of a frame instance are specified a priori by the designer. However, the offset, which identifies the transmitting time of the instance of the frame, has to be calculated satisfying a set of constraints by the schedule synthesizer tool. A cyclic scheduling of all the TT frames, with a *hyper-period*, is defined by the least common multiple of all the TT frames periods [25].

The resolution of the hyper-period is defined by the resolution of the TTEthernet synchronization protocol clock tick, which produces a large range of possible offsets and therefore a high complexity for the schedule synthesizer. One of the most used techniques to reduce its complexity is the definition of a hyper-period with equally sized slots or *raster* [26]. The size chosen for the raster is the maximum time needed to transmit the longest instance of a frame through any dataflow link plus the synchronization precision. Thus, the offset range is reduced to the number of slots in the hyper-period.

B. Scheduling Constraints

TT frames can be formulated by a series of constraints in regards of their offset values assignments of all the network dataflow links. In the following paragraphs we will briefly discuss the constraints implemented in previous synthesizers [9], which are also implemented in this paper, and explain an extension on the application constraints.

a) Avoid-collision constraints: In time-triggered communication only one frame can be transmitted at the same time for a given dataflow link. Hence, the scheduler needs to assign different offset values to different frames.

b) Ensure-causality constraints: A frame being transmitted from a sender to its receivers through a path needs to have the correct sequence of offsets values assigned. In order to control the correct sequence of the frames through the path, a switch can only relay a frame if it has been received and after waiting a predefined number of time slots since it has started receiving the frame, which is typically defined by one time slot.

c) Avoid-buffer-overflow constraints: One of the most important hardware limitations in time-triggered multihop networks is the memory of the switches as frames are stored in their buffers waiting to be transmitted. To control the number of frames that are stored at the same time and to avoid discarding any frame, the avoid-buffer-overflow constraints set the number of time slots that a frame can be waiting in the switch. In this way there are a bounded number of frames that can be at a switch at the same time.

d) Simultaneous-relay constraints: The simultaneous-relay constraints are required for some TTEthernet applications. For a given frame in a switch that has to be relayed through more than one dataflow links, the frame will be dispatched at the same time.

e) Application Constraints: Task level requirements can produce dependencies between different frames such as application constraints in which a frame cannot be sent until a predefined amount of time after the transmission of another frame. These constraints are introduced to facilitate the integration between network and processor level schedules. We differentiate two different application constraints: 1) tight, the amount of time that the successor has to wait after the predecessor is small and it should be dispatched soon after; 2) loose, the amount of time between the predecessor and the successor is large and there is no need to be dispatched as soon as possible. In a factory machine network example, tight constraints are between frames on the same machine, meanwhile loose constraints are between frames of different machines.

IV. SMT-BASED SYNTHESIS

SMT solvers are capable of finding satisfiable instances for a large number of constraints; this feature can be used for synthesize schedules. Frame constraints are asserted into the logical context of the solver which is checked to evaluate if the constraints are satisfiable or not. If they are satisfiable, a model is returned with an instance of a valid schedule.

A. Incremental SMT-based Synthesis

The incremental approach have been introduced as a successful method to synthesize schedules up to 1,000 frames for time-triggered multihop networks [9]. We will only present a brief explanation needed to understand the next approaches. The incremental approach performs different phases until no more frames have to be scheduled:

1) Divide the frames into intervals.
2) Create the frame constraints of the interval.
3) Add the constraints into the logic context.
4) Check the satisfiability of the logic context.
5) If it is satisfiable, push the context and place the satisfiable solution into constraints. Go to step 2.
6) It it is not satisfiable, pop the context. Go to step 3.

The code of the incremental approach can be seen in Listing 1.

Listing 1: Incremental synthesizer

```
1  public void incremental_scheduler() {
2    int head = 1;
3    int tail = stepsize;
4    context_t ctx = yices_new_context();
5    model_t model;
6    init_constraints(head, MAX_NUMBER_FRAMES);
7    while (tail < MAX_NUMBER_FRAMES + 1) {
8      assert_contraints(head, tail);
9      if (yices_check_context(ctx) == STATUS_SAT) {
10       model = yices_get_model();
11       place(model, head, tail);
12       yices_push(ctx);
13       head = tail + 1;
14       tail = tail + stepsize;
15     } else {
16       if (tail - head > threshold) {
17         abort();
18       }
19       yices_pop(ctx);
20       head = head - stepsize;
21     }
22   }
23 }
```

The interval of frames that will be scheduled at each step is defined by $head$ and $tail$, with $stepsize$ interval size. Before executing the loop, the variables that will contain the values of the offset in the logical context are defined with the function $init\_constraints(head, max\_number\_frames)$. For each interval of frames, the constraints are asserted into the logical context with $assert\_constraints(head, tail)$ and checked executing the $yices\_check\_context(ctx)$ function. If the asserted constraints are satisfiable, the offset values are retrieved from the model and placed as new constraints with $place(model, head, tail)$. Last, the context is saved for backtracking purposes using the $yices\_push(ctx)$ function and the indeces are updated. In case no satisfiable solution can be found, $yices\_pop(ctx)$ will backtrack the context, and the $head$ index is reduced in order to schedule the now unplaced frames. If the algorithm has no more contexts to backtrack, the synthesizer will abort and no schedule will be returned.

B. Simple Decomposition SMT-based Synthesis

The number of constraints needed to synthesize schedules of time-triggered multihop networks is large, e.g. a network with a snowflake topology with 50 dataflow links and 1,000 frames generates a total number of $6 \cdot 10^6$ constraints that the SMT solver has to compute at the logical context. The incremental approach removes some of these constraints, but it still lacks scalability when the number of constraints is in the order of 10^7 or more.

We have developed a simple decomposition synthesizer, that without the implementation of application constraints, solves the constraints in different logical contexts enabling a much better scalability than the incremental approach. Hence, we decompose the total number of frames in subsets that are solved in a different logical context at a time. Application constraints have been implemented in the next approach (Section IV-C). Every subset is scheduled sequentially with the previous discussed incremental approach. If the solver returns a satisfiable flag for a given subset, the frame's offsets are stored into internal memory and the logical context of the

solver is reset to remove all the asserted constraints. The loop is executed until all subsets are scheduled and the synthesizer output is a schedule for all the frames of the network. In the case the solver returns an unsatisfiable flag for one subset, the synthesizer will stop and no schedule can be found. The only case that can lead the synthesizer to stop without a solution is when the schedule in construction has already a larger hyper-period than the maximum provided in the input parameters. The decomposition synthesizer algorithm can be found in Listing 2.

Listing 2: Decomposition synthesizer

```
1  public void decomposition_scheduler() {
2    int head = 1;
3    int tail = stepsize;
4    int hyperhead = 1;
5    int hypertail = hyperstepsize;
6    int interval = 0;
7    context_t ctx = yices_new_context();
8    model_t model;
9    while (hypertail < MAX_NUMBER_FRAMES + 1) {
10     init_constraints(hyperhead, hypertail);
11     while (tail < hypertail + 1){
12       assert_contraints(head, tail);
13       if (yices_check_context(ctx) == STATUS_SAT){
14         model = yices_get_model();
15         place(model, head, tail);
16         yices_push(ctx);
17         head = tail + 1;
18         tail = tail + stepsize;
19       } else {
20         if (tail - head > hyperstepsize) {
21           abort();
22         }
23         yices_pop(ctx);
24         head = head - STEPSIZE;
25       }
26     }
27     memory = save_model_into_memory(model);
28     maximum_offset = study_maximum_offset(memory);
29     hyperhead = hypertail + 1;
30     hypertail = hypertail + hyperstepsize;
31     head = hyperhead;
32     tail = head + stepsize - 1;
33     yices_reset_context(ctx);
34   }
35 }
```

The main difference between the simple decomposition synthesizer and the incremental synthesizer is the implementation of a new loop for every subset of frames. The range of the subset is defined by $hyperhead$ and $hypertail$ with $hyperstepsize$ number of frames. The $hyperstepsize$ length is set to 100 frames, which is the larger number of frames capable to schedule the incremental synthesizer without presenting saturation, a more detailed study can be found in Section V-B. Every subset of frames has its own logical context and it is solved with the same approach as the incremental scheduler, but with an adaptation. The difference is the $init\_constraints()$ function, which only initialize the constraints of the subset of frames to be scheduled. Once the incremental approach has finished, if a valid schedule has been found, it is saved into memory with the function $memory = save\_model\_into\_memory(model)$ on top of the previous scheduled subsets and a study of the new maximum offset is performed with the $maximum\_offset = study\_maximum\_offset(memory)$ function. Figure 2 shows an example where the new obtained

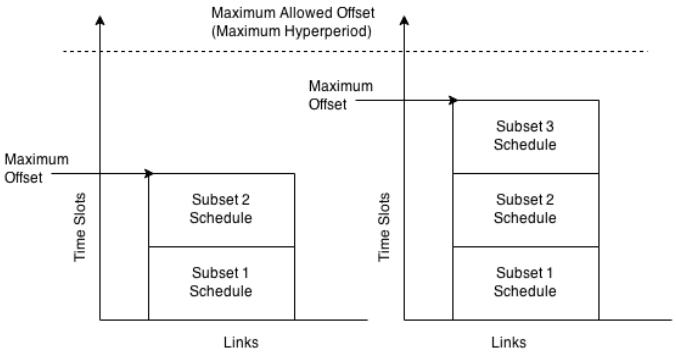

Fig. 2: Example of the $save\_model\_into\_memory()$ and $study\_maximum\_offset()$ functions

schedule subset 3, is saved on top of the schedule subset 2 and the maximum offset of the schedule is updated to its new value. In order to proceed to schedule the next subset, the indeces are updated and the logical context is cleaned with the function $yices\_reset\_context(ctx)$. In case no valid schedule has been found for a subset, the $abort()$ function will be called and the synthesizer will stop without a schedule. If all the subsets have been successfully scheduled, the stacked schedules saved into memory represent the whole network schedule.

Such a simple decomposition is possible as there is no need to define constraints between frames of different subsets. As the application constraints are not implemented in the simple decomposition approach, the only constraints with dependencies are the avoid-collision constraints which, as mentioned before, prevents two instances of different frames from being transmitted at the same time through the same link. However, as every subset of frames is placed in different time slots intervals (Figure 2), there is no need to define such constraints between frames belonging to different groups because they will never be broken.

The decomposition approach presents two main drawbacks compared with the incremental approach. One drawbacks affects the "quality" or "compression" of the hyper-period; scheduling frames on different subsets prevents the synthesizer to continue searching into already scheduled subsets, which could have free slots for more frames to be scheduled. This property does not happen in the incremental approach, as the synthesizer always looks into all the hyper-period space to find free slots for new frames. The other drawback is in regards of the possibility of implementing some new types of constraints into the synthesizer. As it has been explained in the previous paragraph, the simple decomposition approach can be used because there are not dependent constraints between different subsets of frames besides avoid-collision constraints. However in the case there is a need to implement a new type of constraint such as the application constraint (see Section III-B), the simple decomposition synthesizer cannot remember the constraint between different subsets, as they will be "forgotten" after the context reset and the synthesizer will return schedules that probably will break some application constraints.

C. Dependency Sequencing in the Decomposition SMT-based Synthesis

Dependencies between subsets of frames are forgotten after resetting the context, however, to "remember" them alone is not the solution, as there is no backtracking mechanism between subsets of frames like in the incremental synthesizer for the intervals. It could be possible to assert these constraints between different subsets into all the logical contexts and to implement a similar backtracking approach as the incremental synthesizer on the decomposition synthesizer, backtracking full subsets of frames. However the main drawback of the backtracking is the increment of frames per subset to be scheduled every time a break in a constraint is found. In the worst case, backtracking will lead to a subset of all networks frame, which it is impossible to solve in a reasonable amount of time.

The assignment of the frame offsets is strongly dependent on the scheduling frames sequence and it is the cause that some constraints are broken for already scheduled subsets of frames [9]. Building a controlled sequence of frames to be scheduled can avoid these situations, as no constraints will be broken in consequence of previous scheduled frames that had to be scheduled after the actual ones. We propose to implement a dependency tree that will contain all the dependencies between frames of the network which will make us able to know which frames have to be scheduled at every subset of frames in order to avoid the need of a backtracking approach between subsets. The controlled sequencing does not only affect the dependencies of frames on different subsets, it also can group frames whose dependency constraints are tight and can be solved in the same context. The code added to the decomposition approach to implement the dependency tree is shown in the Listings 3.

Listing 3: Dependency ordering in the Decomposition synthesizer

```
1                              ⋮
2  model_t model;
3  int order_frames[hyperstepsize];
4  tree dependency_tree = create_tree();
5  while (hypertail < MAX_NUMBER_FRAMES + 1) {
6    update_tree(dependency_tree);
7    order_frames =
         add_frames_to_schedule(dependency_tree);
8    init_constraints(hyperhead, hypertail);
9    add_intra_dependent(dependencty_tree);
10   while (tail < hypertail + 1){
11                             ⋮
```

Before starting to execute the principal loop, the dependency tree is created with the information of all the dependencies between frames, $create\_tree()$. Inside the loop, for every iteration, $update\_tree(dependencty\_tree)$ remove the already scheduled frames on the previous iteration of the loop and updates the dependency time slots of the remaining frames dependencies. For example, in Figure 3, the top three trees are created by the $create\_tree()$ function and represents the dependencies of the network, in which the numbers inside the nodes are the numbers of the frames, and the numbers on the links are the minimum time slots that the successor nodes has to wait to be sent. In Figure 3, an example of the $update\_tree(dependency\_tree)$ function is presented. The three bottom trees represent the result of their update after the first execution of the main loop. In the first step, frame

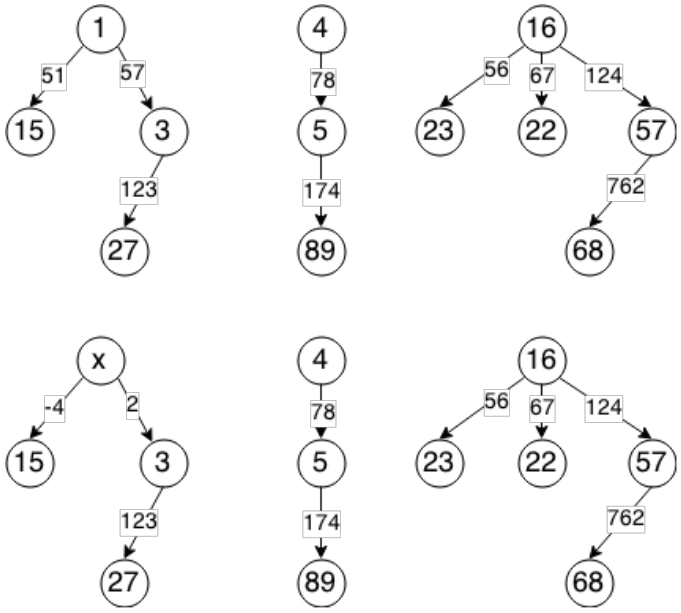

Fig. 3: Example of the $update\_tree(dependencty\_tree, model)$ function for application constraints dependency tree

f_1 is marked as x because it was chosen to be scheduled on the previous subset of frames. In the second step, the links to the successors of f_1 are updated in regards to the offset value of the now scheduled f_1 and the $maximum\_offset$. If the offset value $f_1 = 45$ and $maximum\_offset = 100$, 55 time slots will be deducted on the successors links to f_{15} and f_3. In this way, when a dependency time unit on the tree is smaller than 1 or has no predecessor, the frame can be chosen to be scheduled in the next subset with the $order\_frame = add\_frames\_to\_schedule(dependency\_tree)$ function as there is no possibility to break the dependency constraint. For example, if we wrongly try to schedule f_3 on the next principal loop iteration subset, there is a possibility that its offsets value will be $f_3 = 101$. The top tree in Figure 3 shows that the minimum time slot has to be 57 more than f_1, but its offset is only 56 more resulting in a invalid global schedule. The general case of $update\_tree(dependency\_tree)$ is divided on two steps: first, it removes the already scheduled frames and second, it updates the link numbers of successors of already scheduled frames.

Other dependencies constraints are tight between two frames and can be solved in the same subset of frames with the incremental approach, the $add\_intra\_dependent(dependency\_tree)$ function asserts the constraints to the logical context to be taken into account. Note that dependencies of frames between different subsets are not defined on the logical context, as with the controlled sequencing of the frames we assure that such definition is not needed.

V. Evaluation

A. Test Case Description

In order to evaluate the performance of the decomposition synthesizer with dependency sequencing, we have defined synthetic test cases networks. We have evaluated the synthesizer in four different networks with two different topologies: tree and

snowflake, both with a medium and a large size version. The number of nodes, switches and links are presented in Table I.

	M. Tree	L. Tree	M. Snowflake	L. Snowflake
End Systems	16	64	27	243
Switches	15	63	13	121
Links	30	126	39	363

TABLE I: Number of end systems, switches and links of the test cases networks

The network frames contain one sender and multiple receivers. Different configurations of receivers can be configured, but in our case we defined all the frames as broadcast, meaning that for a given end system sender, the receivers will be all the remaining end systems of the network. We chose broadcast frames as they are the hardest to schedule as they produce the largest number of constraints. Frames have dependencies with other frames in terns of application constraints that will prevent a frame to be sent before a certain time slots of its predecessor. Application constrains are modeled as a tree, in which a frame can have up to five successor frames, but all frames will not have more than one predecessor. The number of frames with application constraints will vary in our test cases on the range of $[0\% - 50\%]$ of the total frames of the network. In all test cases, 20% of the application constraints will be tight, and 80% will be loose. An application constraint will be considered as tight if the successor minimum time slot is smaller than 50 time slots, and loose if is larger.

As for the configuration parameters, we implemented the decomposition synthesizer with Yices 2.3 [27] applying the *Linear Integer Arithmetic* Background Theory [28]. The number of frames to be checked at each interval step in the incremental approach is set at $stepsize = 9$ and the number of frames per subset at $hyperstepsize = 100$. The experiments were run in a MacBook Pro in OS X Yosemite with a 2,6 Ghz Intel Core i7 and 16 GB of RAM.

B. Simple Decomposition SMT-based Synthesis Results

Results of the decomposition synthesizer scheduling for different networks with a number of frames between 1,000 and 100,000 without application constraints can be seen in Figure 4. Note that while the difference in synthesis time between the medium and large snowflake topology networks is small, this is not the case for the medium and large tree topology networks. The increase of the number of end systems and switches in the network does not directly increase the complexity of synthesizing its schedule, as it happens in the snowflake networks. The increase of complexity is due to longer paths to send one frame from the sender to its receivers, which increases the number of constraints to define a frame. This is the reason the large tree network duplicate the needed time to synthesize its schedule as its paths increase much more for larger networks compared with the snowflake topology.

The number of frames in every subset to be scheduled affects the synthesis time needed as the smaller the exponential problems are, the faster that will be solved. For example in Figure 5 for the medium snowflake network in which we change the $hyperstepsize$ value. But there is a drawback in reducing the synthesis time decomposing the schedule in smaller subset of frames; the quality of the schedule is lowered

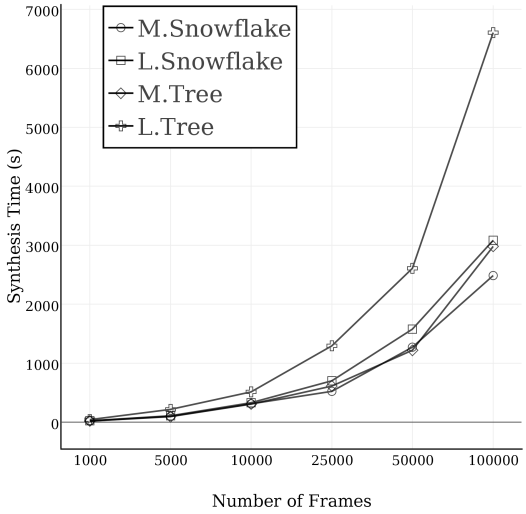

Fig. 4: Synthesis time in seconds of the decomposition approach

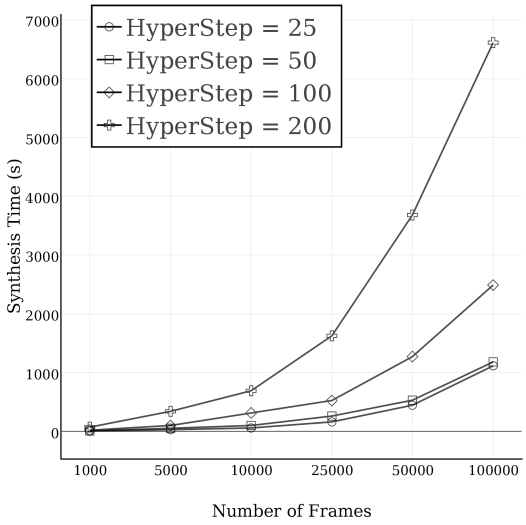

Fig. 5: Synthesis time in seconds of the decomposition approach for different number of frames per subset

as it increases the total time slots needed for the hyper-period as Figure 6 shows. The reason that the hyper-period is increased when the subset of frames is smaller is because every scheduled subset is locked for other frames to be scheduled inside. This prohibits the synthesizer to try to fit more frames into the interval, which has a higher possibility to have free time units for other frames to be scheduled into.

C. Dependency Ordering Decomposition SMT-based Synthesis Results

The modification of the decomposition approach to insert a dependency ordering allows us to synthesize schedules with

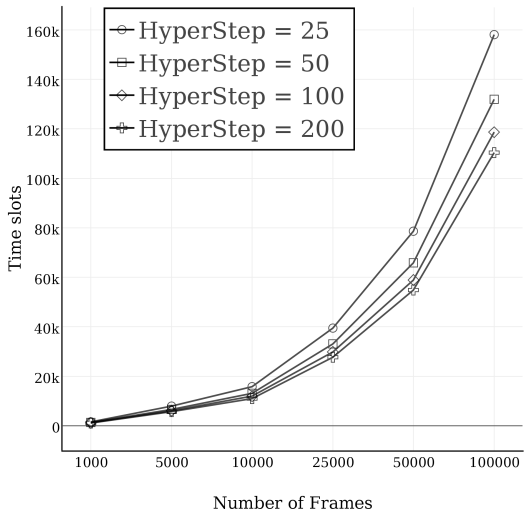

Fig. 6: Minimum hyper-period size in time slots in regards to the number of frames per subset

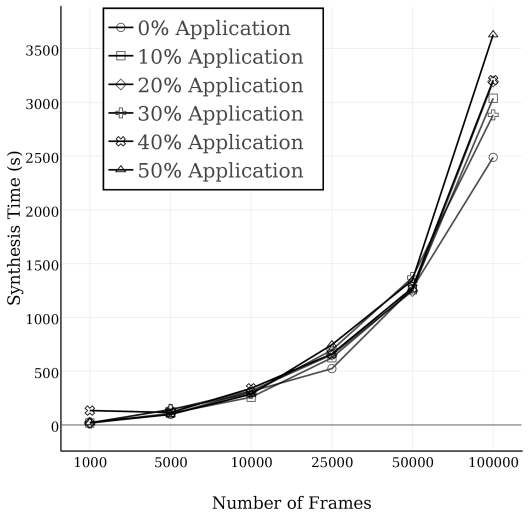

Fig. 7: Synthesis time in seconds of the dependency ordering decomposition approach with application constraints in M. Snowflake

0% to 50% frames with application constraints, in which 20% of them are tight and the rest are loose. Results for the size snowflake network can be seen in Figure 7. The synthesis time decrease compared the incremental synthesizer is about three orders of magnitude, as it takes 1300 seconds to synthesize the schedule of the medium snowflake network with 1,000 frames and 10% frames with application constraints, meanwhile the decomposition synthesizer only takes 19 seconds. The improvement of the performance on the new synthesizer allows us to increase the number of frames to be scheduled, as with the same amount of time we can synthesize schedules of networks up to 50,000 frames. Such improvement is accomplished by linearizing the complexity of an exponential problem with the decomposition of a huge exponential problem in a linear series of small exponential problems, allowing us to synthesize schedules up to 100,000 frames in an hour. In regards to the memory consumption of the decomposition synthesizer, the memory needed is always up to 250 MB independently of the number of frames to be scheduled or the size of the network. This is also an improvement against the incremental synthesizer as the memory consumption was highly related to the number of frames, having a memory consumption up to 4.6 GB to synthesize the schedule of a 1,000 frames network.

The synthesis time for networks without application constraints and the simple decomposition approach (Figure 4) is the same, as the new approach only adds a negligible overhead managing the dependency tree. Networks with different number of application constraints have similar synthesis time as no more constraints are added on the logical context for the loose constraints. Differences are due to tight constraints that are not ordered inside the subset of frames, which are solved with the incremental approach, and are highly dependent of their assertion sequence into the logical context. This can be seen in the 100,000 frames network, when the number of tight application constraints is larger it is more likely that the synthesizer performs backtracking on the schedule of a subset of frames to find a valid schedule.

Networks with longer paths are more prone to break some constraints and consequently the synthesizer has to perform backtracking more frequently than in smaller path networks. Figure 8 shows the synthesis time for the large snowflake network in which more variation are found for networks with more than 25,000 frames. Performing backtracking can greatly increase the synthesis time. For example, the synthesis time for the 10,000 frames network and 30% application constraints is 40% larger than other networks with the same number of frames. In Figure 7, synthesis time for the 1,000 frames network and 40% application constraints is 6 times larger than other same frames networks.

VI. Conclusions and Future Work

In this paper we have presented a decomposition approach to synthesize schedules of the ever-growing time-triggered multihop networks. We decompose the network frames in different subsets and synthesize independent schedules with a state-of-the-art SMT solver. We have also presented a dependency sequencing of the frames to be scheduled to integrate loose application constraints on the decomposition approach with negligible overhead and tight application constraints.

We performed an evaluation of the decomposition approach for different network topologies and sizes that shows an improvement of the synthesis time with three orders of magnitude compared to state-of-the-art approaches, allowing us to synthesize schedules with 50 times more frames in the same amount of time and up to 100,000 frames in one hour. Implementing dependency sequencing provides the possibility to synthesize schedules with up to 50% frames with application constraints. Negligible overhead is added in the synthesis time for the loose constraints. However, an appreciable and unpredictable overhead is caused by the tight constraints that becomes larger when the paths in the network grow.

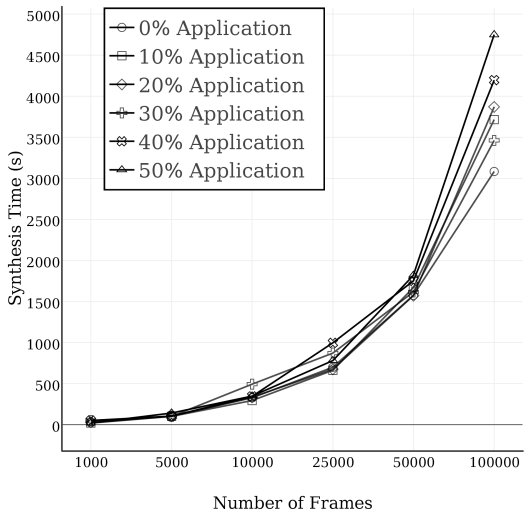

Fig. 8: Synthesis time in seconds of the dependency ordering decomposition approach with application constraints in L. Snowflake

In the evaluation we showed that the hyper-periods could grow to a large number of time slots, which could cause some frames to miss their deadlines, as they have to wait for the execution on the next hyper-period. For future work, we will study scheduling of more than one instance of frames on the same hyper-period for frames with deadlines smaller than the execution of one hyper-period.

ACKNOWLEDGMENT

The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement n°607727 and from the Swedish Knowledge Foundation (KKS), under project n° 20130048.

REFERENCES

[1] H. Kopetz and G. Bauer, "The Time-Triggered Architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[2] M. R. Garey and D. S. Johnson, "Computers and Intractability: a Guide to the Theory of NP-completeness. 1979," *San Francisco, LA: Freeman*, 1979.

[3] J.-P. Watson and J. C. Beck, "A Hybrid Constraint Programming/Local Search Approach to the Job-Shop Scheduling Problem," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2008, pp. 263–277.

[4] C. Fang and L. Wang, "An Effective Shuffled Frog-Leaping Algorithm for Resource-Constrained Project Scheduling Problem," *Computers & Operations Research*, vol. 39, no. 5, pp. 890–901, 2012.

[5] A. Ling, D. P. Singh, and S. D. Brown, "FPGA Logic Synthesis using Quantified Boolean Satisfiability," in *Theory and Applications of Satisfiability Testing*. Springer, 2005, pp. 444–450.

[6] L. De Moura and N. Bjørner, "Satisfiability Modulo Theories: Introduction and Applications," *Communications of the ACM*, vol. 54, no. 9, pp. 69–77, 2011.

[7] B. Dutertre and L. De Moura, "The Yices SMT Solver," *Tool paper at http://yices. csl. sri. com/tool-paper. pdf*, vol. 2, p. 2, 2006.

[8] L. De Moura and N. Bjørner, "Z3: An efficient SMT Solver," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[9] W. Steiner, "An Evaluation of SMT-based Schedule Synthesis for Time-Triggered Multi-Hop Networks," in *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*. IEEE, 2010, pp. 375–384.

[10] "TTEthernet Specification," *TTTech Computertechnik AG, Nov*, 2008.

[11] L. Cordeiro, B. Fischer, and J. Marques-Silva, "SMT-based Bounded Model Checking for Embedded ansi-c Software," *Software Engineering, IEEE Transactions on*, vol. 38, no. 4, pp. 957–974, 2012.

[12] G. Li and G. Gopalakrishnan, "Scalable SMT-based Verification of GPU Kernel Functions," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2010, pp. 187–196.

[13] J. Peleska, E. Vorobev, and F. Lapschies, "Automated Test Case Generation with SMT-solving and Abstract Interpretation," in *NASA Formal Methods*. Springer, 2011, pp. 298–312.

[14] S. Gulwani, S. Jha, A. Tiwari, and R. Venkatesan, "Synthesis of Loop-Free Programs," in *ACM SIGPLAN Notices*, vol. 46, no. 6. ACM, 2011, pp. 62–73.

[15] S. S. Craciunas and R. S. Oliver, "SMT-based Task-and Network-level Static Schedule Generation for Time-Triggered Networked Systems," in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*. ACM, 2014, p. 45.

[16] J. Huang, J. O. Blech, A. Raabe, C. Buckl, and A. Knoll, "Static Scheduling of a Time-Triggered Network-on-Chip based on SMT Solving," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2012, pp. 509–514.

[17] Z. Hanzálek, P. Burget, and P. Sucha, "Profinet IO IRT Message Scheduling," in *Real-Time Systems, 2009. ECRTS'09. 21st Euromicro Conference on*. IEEE, 2009, pp. 57–65.

[18] A. Metzner, M. Franzle, C. Herde, and I. Stierand, "Scheduling Distributed Real-Time Systems by Satisfiability Checking," in *Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on*. IEEE, 2005, pp. 409–415.

[19] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task- and Network-Level Schedule Co-synthesis of Ethernet-based Time-Triggered Systems." in *ASP-DAC*, 2014, pp. 119–124.

[20] D. Tamas-Selicean, S. Marinescu, and P. Pop, "Analysis and Optimization of Mixed-Criticality Applications on Partitioned Distributed Architectures," 2012.

[21] Y. Shin and K. Choi, "Software Synthesis through Task Decomposition by Dependency Analysis," in *Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on*. IEEE, 1996, pp. 98–102.

[22] I. Harjunkoski and I. E. Grossmann, "A Decomposition Approach for the Scheduling of a Steel Plant Production," *Computers & Chemical Engineering*, vol. 25, no. 11, pp. 1647–1660, 2001.

[23] S. Choi and K. Wang, "Flexible Flow Shop Scheduling with Stochastic Processing Times: A Decomposition-based Approach," *Computers & Industrial Engineering*, vol. 63, no. 2, pp. 362–373, 2012.

[24] *ARINC specification 664P7, Aircraft Data Network, Part 7, Avionics Full Duplex Switched Ethernet (AFDX) Network*, Aeronautical Radio Inc., 2005.

[25] T. P. Baker and A. Shaw, "The Cyclic Executive Model and Ada," *Real-Time Systems*, vol. 1, no. 1, pp. 7–25, 1989.

[26] A. K. Mok and W. Wang, "Window-Constrained Real-Time Periodic Task Scheduling," in *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*. IEEE, 2001, pp. 15–24.

[27] B. Dutertre, "Yices 2.2," in *Computer Aided Verification*. Springer, 2014, pp. 737–744.

[28] S. Ranise and C. Tinelli, "Satisfiability Modulo Theories," *Trends and Controversies-IEEE Intelligent Systems Magazine*, vol. 21, no. 6, pp. 71–81, 2006.